Connectivity Labeling and Routing with Multiple Vertex Failures*

Merav Parter
Weizmann Institute of Science
Rehovot, Israel
merav.parter@weizmann.ac.il

Asaf Petruschka
Weizmann Institute of Science
Rehovot, Israel
asaf.petruschka@weizmann.ac.il

Seth Pettie
University of Michigan
Ann Arbor, USA
pettie@umich.edu

ABSTRACT

We present succinct *labeling schemes* for answering connectivity queries in graphs subject to a specified number of *vertex failures*. An f-vertex/edge fault tolerant (f-V/EFT) connectivity labeling is a scheme that produces succinct labels for the vertices (and possibly to the edges) of an n-vertex graph G, such that given only the labels of two vertices s, t and of at most f faulty vertices/edges F, one can infer if s and t are connected in $G - F$. The primary complexity measure is the maximum label length (in bits).

The f-EFT setting is relatively well understood: [Dory and Parter, PODC 2021] gave a randomized scheme with succinct labels of $O(\log^3 n)$ bits, which was subsequently derandomized by [Izumi et al., PODC 2023] with $\tilde{O}(f^2)$-bit labels. As both noted, handling vertex faults is more challenging. The known bounds for the f-VFT setting are far away: [Parter and Petruschka, DISC 2022] gave $\tilde{O}(n^{1-1/2^{\Theta(f)}})$-bit labels, which is linear in n already for $f = \Omega(\log \log n)$.

In this work we present an efficient f-VFT connectivity labeling scheme using $\text{poly}(f, \log n)$ bits. Specifically, we present a randomized scheme with $O(f^3 \log^5 n)$-bit labels, and a derandomized version with $O(f^7 \log^{13} n)$-bit labels, compared to an $\Omega(f)$-bit lower bound on the required label length. Our schemes are based on a new *low-degree graph decomposition* that improves on [Duan and Pettie, SODA 2017], and facilitates its distributed representation into labels. This is accompanied with specialized *linear graph sketches* that extend the techniques of the Dory and Parter to the vertex fault setting, which are derandomized by adapting the approach of Izumi et al. and combining it with *hit-miss hash families* of [Karthik and Parter, SODA 2021].

Finally, we show that our labels naturally yield routing schemes avoiding a given set of at most f vertex failures with table and header sizes of only $\text{poly}(f, \log n)$ bits. This improves significantly over the linear size bounds implied by the EFT routing scheme of Dory and Parter.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; **Data structures design and analysis**; **Distributed algorithms**.

*Supported by NSF Grant CCF-2221980, by the European Research Council (ERC) under the European Union's Horizon 2020 Research and Innovation Programme, Grant Agreement No. 949083, and by the Israel Science Foundation (ISF), Grant 2084/18.

STOC '24, June 24–28, 2024, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0383-6/24/06
https://doi.org/10.1145/3618260.3649729

KEYWORDS

Labeling Schemes, Routing Schemes, Fault Tolerance

ACM Reference Format:
Merav Parter, Asaf Petruschka, and Seth Pettie. 2024. Connectivity Labeling and Routing with Multiple Vertex Failures. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC '24), June 24–28, 2024, Vancouver, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3618260.3649729

1 INTRODUCTION

Labeling schemes are fundamental distributed graph data structures, with various applications in communication networks, distributed computing and graph algorithms. Such schemes are concerned with assigning the vertices (and possibly also edges) of a given graph with succinct and meaningful names, or *labels*. The inherent susceptibility to errors in many real-life networks creates a need for supporting various logical structures and services in the presence of failures. The focus of this paper is on labeling and routing schemes for connectivity under a limited number of *vertex faults*, which is poorly understood compared to the edge fault setting.

Let $G = (V, E)$ be an n-vertex graph, and $f \geq 1$ be an integer parameter. An f-vertex fault tolerant (VFT) labeling scheme assigns short labels to the vertices, so that given a query $\langle s, t, F \rangle \in V \times V \times \binom{V}{\leq f}$, one can determine if s and t are connected in $G - F$, merely by inspecting the labels of the query vertices $\{s, t\} \cup F$. Edge fault tolerant (EFT) labelings are defined similarly, only with $F \subseteq E$. The main complexity measure of a labeling scheme is the maximal *label length* (in bits), while construction and query time are secondary.

Since their first explicit introduction by Courcelle and Twigg [9] and until recently, all f-EFT and f-VFT labeling schemes were tailored to specialized graph classes, such as bounded treewidth, planar, or bounded doubling dimension [1, 2, 8, 8, 9], or limited to handling only a small number of faults [6, 20, 26].

Dory and Parter [10] were the first to provide f-EFT connectivity labels for general graphs. They developed a *randomized* scheme with label size of $O(\log^3 n)$ bits, regardless of f, in which queries are answered correctly with high probability, i.e., of $1-1/\text{poly}(n)$. Their construction is based on the *linear graph sketching* technique of [3, 18]. Notably, their labels can be used in an almost black-box manner to yield approximate distances and routing schemes; see [6, 10]. By increasing the label length of the Dory-Parter scheme to $\tilde{O}(f)$ bits, the randomly assigned labels will, with high probability, answer *all* possible $n^{O(f)}$ queries correctly. Izumi, Emek, Wadayama, and Masuzawa [16] provided a full derandomization of the Dory-Parter scheme, where labels are assigned deterministically in polynomial time and have length $\tilde{O}(f^2)$ bits.

Vertex faults are considerably harder to deal with than edge faults. A small number of failing vertices can break the graph into a possibly linear number of connected components. Moreover, known

structural characterization of *how* f vertex faults change connectivity are lacking, unless f is small; see [4, 17, 28, 29]. By a naive reduction from vertex to edge faults, the Dory-Parter scheme yields VFT connectivity labels of size $\tilde{O}(\Delta(G))$, where $\Delta(G)$ is the maximum degree in G. This dependency is unsatisfactory, as $\Delta(G)$ might be even linear in n. Very recently, Parter and Petruschka [26] designed f-VFT connectivity labeling schemes for small values of f. For $f = 1$ and $f = 2$ their labels have size $O(\log n)$ and $O(\log^3 n)$, respectively, and in general the size is $\tilde{O}(n^{1-1/2^{f-2}})$, which is sublinear in n whenever $f = o(\log\log n)$. By comparing this state of affairs to the EFT setting, the following question naturally arises:

Question 1.1. *Is there an f-VFT connectivity labeling scheme with labels of* $\mathrm{poly}(f, \log n)$ *bits?*

Compact Routing. An essential requirement in communication networks is to provide efficient routing protocols, and the error-prone nature of such networks demands that we route messages avoiding vertex/edge faults. A routing scheme consists of two algorithms. The first is a preprocessing algorithm that computes (succinct) routing tables and labels for each vertex. The second is a routing algorithm that routes a message from s to t. Initially the labels of s, t are known to s. At each intermediate node v, upon receiving the message, v uses only its local table and the (short) header of the message to determine the next-hop, specified by a *port number*, to which it should forward the message. When dealing with a given set F of at most f faults, the goal is to route the message along an s-to-t path in $G - F$. We consider the case where the labels of F are initially known to s, also known as *forbidden-set routing*.[1].

The primary efficiency measures of a routing scheme are the *space* of the routing tables, labels and headers; and the *stretch* of the route, i.e., the ratio between the length of the s-t routing path in $G-F$, and the corresponding shortest path distance. In the fault-free and EFT settings, efficient routing schemes for in general graphs are known; we refer to [10] for an overview. The known bounds for VFT routing schemes in general graphs are much worse; there is no such scheme with space bounds sublinear in n, even when allowing unbounded stretch. This is in sharp contrast to the f-EFT setting for which [10] provides each vertex a table of $\tilde{O}(f^3 n^{1/k})$ bits, labels of $\widetilde{O}(f)$ bits (for vertices and edges) and headers of $\tilde{O}(f^3)$ bits, while guaranteeing a route stretch of $O(kf)$. The current large gap in the quality of routing schemes under vertex faults compared to their edge-faulty counterparts leads to the following question.

Question 1.2. *Is there an f-VFT routing scheme for general graphs with* sublinear *space bounds for tables, labels and headers?*

The Centralized Setting and Low-Degree Decompositions. A closely related problem is that of designing *centralized sensitivity oracles* for f-VFT connectivity, which, in contrast to its distributed labeling counterpart, is very well understood. Results of Duan and Pettie [12] followed by Long and Saranurak [22] imply an $\tilde{O}(\min\{m, fn\})$-space data structure (where m is the number of edges), that updates in response to a given failed set $F \subseteq V$, $|F| \le f$ within $\hat{O}(f^2)$ time, then answers connectivity queries in $G - F$ in $O(f)$ time. These bounds are almost-optimal under certain hardness assumptions [15, 21, 22]. See [29, 30] for similar oracles with update/query time independent of n.

As previously noted, a major challenge with vertex faults, arising also in the centralized setting, is dealing with large degrees. To tackle this challenge, Duan and Pettie [12] used a recursive version of the Fürer-Raghavachari [13] algorithm to build a *low-degree hierarchy*. For any graph G, it returns a $\log n$-height hierarchical partition of $V(G)$ into vertex sets, each spanned by a Steiner tree of degree at most 4. For f-VFT connectivity queries, having an $O(1)$-degree tree is almost as good as having $\Delta(G) = O(1)$. Duan, Gu, and Ren [11] extended the low-degree hierarchy [12] to answer f-VFT approximate distance queries, and Long and Saranurak [22] gave a faster construction of low-degree hierarchies (with $n^{o(1)}$-degree trees) using expander decompositions. However, prior usages of such hierarchies seem to hinge significantly on centralization, and facilitating their distributed representation for labeling schemes calls for new ideas.

Our Results. The central contribution of this paper is in settling Question 1.1 to the affirmative. We present new randomized and deterministic labeling schemes for answering f-failure connectivity queries with label length $\mathrm{poly}(f, \log n)$, which improves on [26] for all $f \ge 3$. Our main result is:

Theorem 1.1. *There is a randomized polynomial-time labeling scheme for f-VFT connectivity queries that outputs labels with length $O(f^3 \log^5 n)$. That is, the algorithm computes a labeling function $L : V \to \{0,1\}^{O(f^3 \log^5 n)}$ such that given $L(s)$, $L(t)$ and $\{L(v) \mid v \in F\}$ where $|F| \le f$, one can report whether s and t are connected in $G - F$, which is correct with probability $1 - 1/\mathrm{poly}(n)$.*

This resolves an open problem raised in [10], improves significantly over the state-of-the-art $\mathrm{poly}(n)$-bit labels when $f \ge 3$ [10, 26], and is only polynomially off from an $\Omega(f)$-bit lower bound (see the full version [27]). The labeling scheme of Theorem 1.1 is based on a new low-degree hierarchy theorem extending the Duan-Pettie [12] construction, which overcomes the hurdles presented by the latter for facilitating its distributed representation.

Further, in the full version [27] we derandomize the construction of Theorem 1.1, by combining the approach of Izumi et al. [16] with the deterministic "hit-miss hashing" technique of Karthik and Parter [19], which addresses an open problem of Izumi et al. [16].

Theorem 1.2. *There is a deterministic polynomial-time labeling scheme for f-VFT connectivity queries that outputs labels with length $O(f^7 \log^{13} n)$.*

To address Question 1.2, we use the labels of Theorem 1.1 that naturally yield compact routing schemes in the presence of f vertex faults. In the full version [27] we show:

Theorem 1.3. *There is a randomized forbidden-set routing scheme resilient to f (or less) vertex faults, that assigns each vertex $v \in V$ a label $L(v)$ of $O(f^3 \log^5 n)$ bits, and a routing table $R(v)$ of $O(f \log n)$ bits. The header size required for routing a message is $O(f \log^2 n)$ bits. The s-t route has $O(fn \log n)$ many hops.*

[1] This assumption is made only for simplicity and clarity of presentation. It can be omitted at the cost of increasing the route length and the space bounds by factors that are small polynomials in f, using similar ideas as in [10].

This improves considerably upon the current linear space bounds implied by the f-EFT routing scheme of [10], with the same hop-bound. The routing scheme can also be derandomized in a straightforward manner, using the deterministic labels of Theorem 1.2.

Preliminaries. Throughout, we fix the n-vertex input graph $G = (V, E)$, assumed to be connected without loss of generality. For $U \subseteq V$, $G[U]$ and $G - U$ denote the subgraphs of G induced by U and $V - U$, respectively. When P_0, P_1 are paths, $P_0 \circ P_1$ denotes their concatenation, defined only when the last vertex of P_0 coincides with the first vertex of P_1. We use the operator $\oplus$ to denote *both* the symmetric difference of sets ($A \oplus B = (A - B) \cup (B - A)$) and the bitwise-XOR of bit-strings. The correct interpretation will be clear from the type of the arguments.

2 TECHNICAL OVERVIEW

At the macro level, our main f-VFT connectivity labeling scheme (Theorem 1.1), is obtained by substantially extending and combining two main tools: (I) The Dory-Parter [10] labels for connectivity in presence of *edge faults*, based on the *linear graph sketching* technique of [3, 18]. (II) The Duan-Pettie [12] *low-degree hierarchy*, originally constructed for *centralized* connectivity oracles under vertex failures.

We start with a short primer on graph sketching and the Dory-Parter labeling scheme, since we build upon these techniques in a "white box" manner. Our starting observation shows how the Dory-Parter labels can be extended to handle vertex faults, when assuming the existence of a *low-degree spanning tree*. We then introduce the Duan-Pettie low-degree hierarchy, which has been proven useful in the centralized setting; intuitively, such a hierarchy lets us reduce general graphs to the low-degree spanning tree case. We explain our strategy for using a low-degree hierarchy to obtain an f-VFT labeling scheme, which also pinpoints the hurdles preventing us from using the Duan-Pettie hierarchy "as is" for this purpose. Next, we discuss the resolution of these hurdles obtained by novel construction of low-degree hierarchies with improved key properties, and tie everything together to describe the resulting scheme. Finally, we briefly discuss how to optimize the label size by a new combination of graph sketches with graph sparsification and *low-outdegree orientations*.

2.1 Graph Sketches and the Dory-Parter Labels

Graph Sketches. The *linear graph sketching* technique of [3, 18] is a tool for identifying *outgoing edges* from a given vertex subset $U \subseteq V$. We give a short informal description of how it works, which could be skipped by the familiar reader. Generate nested edge-subsets $E = E_0 \supseteq E_1 \supseteq \cdots \supseteq E_{O(\log n)} = \emptyset$ by sampling each $e \in E_i$ into E_{i+1} with probability $1/2$. Thus, for any $\emptyset \neq E' \subseteq E$, some E_i contains *exactly one* of the edges in E', with some constant probability. The sketch of E', denoted $\mathsf{sketch}(E')$, is a list where the i-th entry holds the *bitwise-XOR* of (the identifiers) of edges from E' sampled into E_i: $\bigoplus_{e \in E' \cap E_i} \mathsf{id}(e)$. Crucially, the sketches are *linear* with respect to the $\bigoplus$ operator: $\mathsf{sketch}(E') \oplus \mathsf{sketch}(E'') = \mathsf{sketch}(E' \oplus E'')$. The edge sketches are extended to vertex subsets $U \subseteq V$ as

$$\mathsf{sketch}(U) = \bigoplus_{u \in U} \mathsf{sketch}(\{e \in E \mid e \text{ incident to } u\}).$$

By linearity, the $U \times U$ edges cancel out, so $\mathsf{sketch}(U)$ is the sketch of *outgoing edges from* U. Most entries in $\mathsf{sketch}(U)$ are "garbage strings" formed by XORing many edges, but the sketch property ensures that one of them contains $\mathsf{id}(e)$ of an edge e outgoing from U, with constant probability.

The Dory-Parter Labels. Our approach builds upon the Dory-Parter [10] labels for *edge faults*, which we now briefly explain. Choose any rooted spanning tree T of G. Construct standard T-*ancestry labels*: each $v \in V$ gets an $O(\log n)$-bit string $\mathsf{anc}(v)$. Given $\mathsf{anc}(u), \mathsf{anc}(v)$ one can check if u is a T-ancestor of v. These are the vertex labels. The label of an edge $e = \{u, v\}$ always stores $\mathsf{sketch}(e)$ and $\mathsf{anc}(u), \mathsf{anc}(v)$. The labels of *tree edges* are the ones doing the heavy lifting: if $e \in E(T)$, we additionally store the *subtree-sketches* $\mathsf{sketch}(V(T_u))$ and $\mathsf{sketch}(V(T_v))$, where T_x denotes the subtree rooted at x.

Given the labels of $s, t \in V$ and of failing $F \subseteq E$, the connectivity query (i.e., if s, t are connected in $G - F$) is answered by a forest growing approach in the spirit of Borůvka's 1926 algorithm [5, 25]. Letting $F_T = F \cap E(T)$, observe that $T - F_T$ consists of $|F_T| + 1$ connected *parts* $\mathcal{P} = \{P_0, \ldots, P_{|F_T|}\}$. Each part can be expressed as $P_i = \bigoplus_x V(T_x)$, where the $\bigoplus$ runs over some subset of endpoints of F_T. Thus, at initialization, the algorithm computes the sketch $\mathsf{sketch}(P_i)$ by XORing subtree-sketches stored in the F_T-labels. (It knows which subtrees to XOR using the ancestry labels.) To avoid getting outgoing-edges that are in F, the F-edges are *deleted* from the relevant part-sketches: For each $e = \{u, v\} \in F$, we locate the parts $P_u, P_v \in \mathcal{P}$ that contain u, v (using ancestry labels), and if $P_u \neq P_v$, we update the sketches of P_u, P_v by XORing them with $\mathsf{sketch}(e)$. So, the part-sketches now refer to $G - F$ instead of G.

We next run Borůvka, by working in $O(\log n)$ *rounds*. In each round, we use the part-sketches to find outgoing edges and *merge* parts along them, forming a coarser partition. The sketches of the new parts are computed by XORing the old ones. By the final round, the parts become the connected components of $G - F$, with high probability. Finally, we locate which initial parts contained s, t using the ancestry labels, and see if these ended up in the same final part.

2.2 Starting Point: Vertex Faults in Low-Degree Spanning Tree

The intuition for our approach comes from the following idea. Suppose we were somehow able to find a spanning tree T of G with small maximum degree, say $\Delta(T) = \tilde{O}(1)$. Since the *tree edges* are the ones doing the heavy lifting in the Dory-Parter scheme (by storing the subtree-sketches), the label of a failing vertex x may store only the $\tilde{O}(1)$ labels of x's incident edges in T. However, there is an issue: how do we delete the non-tree edges incident to failing vertices from the part-sketches? We cannot afford to store the sketch of each such edge explicitly, as the degrees in G may be high.

To overcome this issue, we use the paradigm of *fault-tolerant sampling*, first introduced by [7, 31]. We generate f^2 random subgraphs $G_1, \ldots, G_{f^2}$. Each G_i is formed by sampling each vertex w.p. $1/f$, and keeping only the edges with both endpoints sampled. This ensures that for every fault-set $F \subseteq V$, $|F| \leq f$, and every edge e of $G - F$, with constant probability, at least one G_i contains e (G_i "hits" e) but no edge incident to F (G_i "misses" F). We replace the

subtree-sketches stored in the labels with f^2 basic sketches, one for each G_i. When trying to get an outgoing edge from a part P, the guarantee is that with constant probability, there will be some basic G_i-sketch of P such that G_i misses F but hits one of the outgoing edges of P in $G-F$; such a basic sketch which will provide us (again with constant probability) a desired outgoing edge.

The label length of the approach above becomes $\tilde{O}(f^2 \cdot \Delta(T))$ bits, as each vertex stores f^2 basic sketches for each of its incident tree edges.

2.3 The Duan-Pettie Low-Degree Hierarchy

The issue with the low-degree spanning tree idea is clear: such a tree might not exist. The *low-degree hierarchy* of Duan and Pettie [12] was designed for centrlized oracles for connectivity under vertex faults, in order to tackle exactly this issue. Their construction is based on a recursive version of the Fürer-Raghavachari algorithm [13], but understanding the algorithm is less important for our current purposes. Rather, we focus on explaining its output, namely, *what the low-degree hierarchy is*, and what are its key properties.

The Duan-Pettie hierarchy[2] $\mathcal{H}^0$ consists of a partition $\mathcal{C}$ of the vertices V into *components*. We use the letter γ to denote one such component. So, $V = \bigcup_{\gamma \in \mathcal{C}} \gamma$, and $\gamma \cap \gamma' = \emptyset$ for any two distinct components $\gamma, \gamma' \in \mathcal{C}$. The components in $\mathcal{C}$ are hierarchically placed as the nodes of a *virtual tree* (hence the name "hierarchy"). We call the virtual hierarchy edges *links*, to distinguish them from the edges of the original graph G. For two components $\gamma, \gamma' \in \mathcal{C}$, we denote $\gamma \prec \gamma'$ if γ is a strict *descendant* of γ' (i.e., γ' is a strict *ancestor* of γ) in the hierarchy tree. Two components γ, γ' such that $\gamma \preceq \gamma'$ or $\gamma \succeq \gamma'$ are called *related*. The key properties of the hierarchy $\mathcal{H}^0$ are as follows:

(1) **Logarithmic height:** The hierarchy tree $\mathcal{H}^0$ has height $O(\log n)$.
(2) **No lateral edges:** There are no *lateral* G-edges that cross between unrelated components. Namely, if $\{u, v\}$ is an edge of G, and $\gamma_u, \gamma_v \in \mathcal{C}$ are the components containing u, v respectively, then γ_u and γ_v are related.
(3) **Connected sub-hierarchies:** The vertices in each sub-hierarchy induce a connected subgraph of G. Namely, let $\mathcal{H}^0_\gamma$ be the subtree of $\mathcal{H}^0$ rooted at component $\gamma \in \mathcal{C}$, and $V(\mathcal{H}^0_\gamma) = \bigcup_{\gamma' \preceq \gamma} \gamma'$ be the vertices appearing in descdedants of γ (i.e., found in the nodes of $\mathcal{H}^0_\gamma$). Then the subgraph $G[V(\mathcal{H}^0_\gamma)]$ is connected.
(4) **Low-degree Steiner trees:** Each component $\gamma \in \mathcal{C}$ is associated with a *Steiner tree* $T^0(\gamma)$, whose *terminal set* is γ. The tree $T^0(\gamma)$ is a subgraph of G that spans all the vertices in γ, and has maximum degree ≤ 4. However, it may contain *Steiner points*: vertices outside γ.

2.4 First Attempt: with Duan-Pettie Hierarchy

We now give an overview of how we would like to use the low-degree hierarchy, by taking the following methodological approach: First, we provide the general idea for constructing labels based on a low-degree hierarchy such as the Duan-Pettie hierarchy $\mathcal{H}^0$. Then, we highlight the key properties that are *missing* from $\mathcal{H}^0$ to make it satisfactory for our purposes. In the following subsection (Section 2.5), we present our modified low-degree hierarchy constructions, which mitigate these barriers.

Preprocessing: Creating the Auxiliary "Shortcuts-Graph" $\hat{G}$. The labels are built on top of an *auxiliary graph* $\hat{G}$ computed in a preprocessing step. The graph $\hat{G}$ consists of all G-edges plus an additional set of *shortcut edges* that are computed based on the hierarchy, as explained next. For a component γ, let $N(\mathcal{H}^0_\gamma)$ denote the set of vertices *outside* $V(\mathcal{H}^0_\gamma)$ that are adjacent to some vertex *inside* $V(\mathcal{H}^0_\gamma)$ (that is, the *neighbors of* $\mathcal{H}^0_\gamma$). Note that as there are no lateral edges, $N(\mathcal{H}^0_\gamma)$ contains only vertices from strict ancestor components of γ. Also, by the connected sub-hierarchies property, every distinct $u, v \in N(\mathcal{H}^0_\gamma)$ are connected in G by a path whose internal vertices are contained in $V(\mathcal{H}^0_\gamma)$. We therefore add a shortcut edge between u, v that represents the existence of such a path. To make sure we know that this edge corresponds to a path through $V(\mathcal{H}^0_\gamma)$, the shortcut edge is marked with *type* "γ". To conclude, the auxiliary graph $\hat{G}$ is the graph formed by starting with G, giving all its edges type "*original*", and then, for each $\gamma \in \mathcal{C}$, adding a clique on $N(\mathcal{H}^0_\gamma)$ with edges of type "γ". Note that there may be multiple edges (with different types) connecting two vertices, so $\hat{G}$ is an *edge-typed multi-graph*.

Query: Affected Components and the Query Graph G^*. We now shift our attention to focus on how any specific connectivity query $\langle s, t, F \rangle$ interacts with $\hat{G}$. First, we define the notion of components that are *affected* by the query. Intuitively, an affected component is one whose corresponding shortcut edges are no longer trusted, because the path they represent might contain faults from F. Formally, $\gamma \in \mathcal{C}$ is called *affected* if $V(\mathcal{H}^0_\gamma) \cap (F \cup \{s, t\}) \neq \emptyset$.[3] Observe that the set of affected components is *upwards-closed*: If γ is affected, then every $\gamma' \succeq \gamma$ is also affected. As the query vertices $F \cup \{s, t\}$ lie only in at most $f + 2$ different components, and the hierarchy has $L \leq \log n$ levels, there are only $O(f \log n)$ affected components. The *query graph* G^* is defined as the subgraph of $\hat{G}$ that consists of all vertices lying in affected components, and all the edges of $\hat{G}$ that connect them and have *unaffected types*. Namely, we delete "*bad*" shortcut edges whose types are affected. The key property we prove about G^* is that s, t are connected in $G - F$ if and only if they are connected in $G^* - F$. Hence, we would like our labels to support Borůvka execution in $G^* - F$. Note that unlike $\hat{G}$, which depends only on G, the graph G^* is a function of G and of the query elements s, t and F. As we will see, one of the challenges of the decoding algorithm will be in performing computation on G^* given label information computed based on the *preprocessing graph* $\hat{G}$.

Key Obstacles in Labelizing the Duan-Pettie Hierarchy. The general idea is that each affected component γ has a low-degree spanning tree $T^0(\gamma)$, enabling us to employ our approach for low-degree spanning trees: store in the label of a vertex the sketches of subtrees

[2] The 0-superscript in the notation $\mathcal{H}^0$ is used since the Duan-Pettie hierarchy serves as the initial point for other hierarchy constructions, introduced in Section 2.5.

[3] In case there are no faults from F in $V(\mathcal{H}^0_\gamma)$, we do not really care if s or t are there; the shortcut edges with type "γ" are still reliable. However, it will be more convenient (although not needed) to assume that s, t are in affected components, hence we also force this condition.

rooted at its tree-neighbors. Thus, for each affected component, we can compute the sketches of the parts into which its tree breaks after the vertex-set F fails.[4] Together, these parts constitute the initial partition for running the Borůvka algorithm in $G^* - F$. However, there are two main obstacles:

(a) **Steiner points.** A vertex x appears only in one component γ_x, but can appear in *many trees* $T^0(\gamma)$ with $\gamma \neq \gamma_x$ *as a Steiner point*. So even though x only has ≤ 4 neighbors in each such $T^0(\gamma)$, the total number of subtree-sketches we need to store in x's label may be large.

(b) **Large $N(\mathcal{H}^0_\gamma)$ sets.** The decoding algorithm is required to obtain sketch information with respect to the query graph G^*. When constructing the label of a vertex x (in the preprocessing step), we think of x as participating in an unknown query, which gives only *partial* information on the future graph G^*: all ancestor components $\gamma \geq \gamma_x$ will be affected. To modify $\hat{G}$-sketches into sketches in the query graph G^*, the shortcut-edges of type "γ" should be deleted from the given sketches. To this end, we would like to store in x's label, for every $\gamma \geq \gamma_x$ and every $v \in N(\mathcal{H}^0_\gamma)$, the sketch of the edges $\hat{E}_\gamma(v)$: edges with type "γ" that are incident to v. This is problematic as the neighbor-set $N(\mathcal{H}^0_\gamma)$ might be too large.

2.5 Resolution: New Low-Degree Hierarchies

To overcome obstacles (a) and (b), we develop a new low-degree decomposition theorem, which essentially shows how we can alter the Duan-Pettie hierarchy $\mathcal{H}^0$ to (a) admit low-degree spanning trees without Steiner points, and (b) to have small neighbor-sets of sub-hierarchies.

The Unify *Procedure.* We start with tackling obstacle (a). The idea is rather intuitive: our issue with the trees $\{T^0(\gamma)\}$ is that they may contain edges connecting two different components. I.e., a problematic edge $e = \{u, v\}$ appearing in $T^\cup = \bigcup_{\gamma \in C} T^0(\gamma)$ is such that $\gamma_u \neq \gamma_v$. To fix e, we want to *unify* γ_u and γ_v into one component. As there are no lateral edges, γ_u and γ_v must be related, say $\gamma_u \succ \gamma_v$. If it happened to be that γ_u is the *parent* of γ_v, then this is easy: we merge γ_u, γ_v into a new component $\gamma_{new} = \gamma_u \cup \gamma_v$, associated with the tree formed by connecting $T^0(\gamma_u), T^0(\gamma_v)$ through e, i.e., $T(\gamma_{new}) = T^0(\gamma_u) \cup \{e\} \cup T^0(\gamma_v)$. The child-components of γ_u, γ_v become children of the unified γ_{new}. However, if γ_u is a further-up ancestor of γ_v, such a unification can cause other issues; it may violate the "no lateral edges" and "connected sub-hierarchies" properties. The reason these issues did not appear for a parent-child pair is that their unification can be seen as a *contraction* of a hierarchy link.

We therefore develop a recursive procedure called Unify, that when asked to unify γ_u and γ_v, returns a *connected* set of hierarchy-nodes that contains γ_u, γ_v. Further, Unify exploits the properties of the low-degree hierarchy to also provide edges through which we can connect the trees $T^0(\gamma)$ of the components γ appearing in this set (while keeping the degrees in the unified tree small). Thus, we can unify them and fix e. By iteratively applying Unify to fix problematic $T^\cup$-edges, we end up with a spanning tree for each component, rather than with a Steiner tree. Further, we prove that this does not increase the maximum tree-degree very much; it grows from 4 to only $O(\log n)$.

Hierarchies Based on "Safe" Subsets of Vertices. To tackle obstacle (b), we exploit the following insight: The low-degree requirement can be relaxed, as long as we ensure that the *failed F-vertices* have low degrees in the trees; the degree of non-failing vertices does not matter. At first sight, this might not seem very helpful, as we do not know in advance which vertices are faulty (namely, we should prepare to *any* possible set F of f vertex faults). In order to deal with this challenge, we randomly partition the vertices V into $f+1$ sets $S_1, \ldots, S_{f+1}$. Each of these sets gets a tailor-made hierarchy $\mathcal{H}(S_i)$ constructed for it (which is still a partition of V). When constructing $\mathcal{H}(S_i)$, we think of S_i as a set of *safe* vertices, that will not fail, and are therefore allowed to have high degrees, while the vertices in $V - S_i$ should remain with small degrees. Note that for every $F \subseteq V$ with $|F| \leq f$, there is some S_i such that $S_i \cap F = \emptyset$; the hierarchy $\mathcal{H}(S_i)$ will be used to handle queries with faulty-set F, so that F-vertices will have low degrees, as needed.

We now give a high-level explanation of how our relaxed degree requirement, allowing large degrees for S_i-vertices, can be used for eliminating large neighbor-sets of sub-hierarchies and obtaining $\mathcal{H}(S_i)$. We set the "large" threshold at $\Theta(f \log n)$. Suppose $\gamma \in C$ is some component with $|N(\mathcal{H}^0_\gamma)| = \Omega(f \log n)$. Our goal is to eliminate this problematic component γ. Again, the trick will be unifications. Because each vertex in $N(\mathcal{H}^0_\gamma)$ has probability $1/(f+1)$ to be an S_i-vertex, with high probability, there is some safe vertex $u \in S_i \cap N(\mathcal{H}^0_\gamma)$. Therefore, there is some G-edge $e = \{u, v\}$ with $\gamma_u \succ \gamma \succeq \gamma_v$. We call Unify asking to unite γ_u with γ_v, through the edge e. As Unify returns connected sets of nodes, the resulting unified component will also include the problematic component γ, and it will be eliminated. On a high level, the reason we may use the edge e for connecting trees is because we are allowed to increase the degree of the safe vertex $u \in S_i$. So, after repeatedly eliminating problematic components, all neighbor-sets of sub-hierarchies have size $O(f \log n)$, and the degree of all vertices in $V - S_i$ (i.e., the unsafe vertices) in the trees remains $O(\log n)$.

The New Hierarchies. To summarize, we get $f+1$ hierarchies $\mathcal{H}(S_1), \ldots, \mathcal{H}(S_{f+1})$, each corresponding to one set from a partition $(S_1, \ldots, S_{f+1})$ of the vertices V. So as not to confuse them with the Duan-Pettie Hierarchy $\mathcal{H}^0$, we denote the partition of V to components *in each hierarchy* $\mathcal{H}(S_i)$ by $\mathcal{K}(S_i)$, and denote components such by the letter K (instead of γ). Now, $\mathcal{H}_K(S_i)$ denotes the sub-hierarchy of $\mathcal{H}(S_i)$ rooted at component $K \in \mathcal{K}(S_i)$, and $N(\mathcal{H}_K(S_i))$ denotes its neighbor-set. Each hierarchy $\mathcal{H}(S_i)$ has the following key properties:

(1) (Old) **Logarithmic height:** As before
(2) (Old) **No lateral edges:** As before.
(3) (Old) **Connected sub-hierarchies:** As before.
(4) (Modified) **Spanning trees with low-degrees of unsafe vertices:** Each $K \in \mathcal{K}(S_i)$ is associated with a tree $T(K)$ which is a subgraph of G containing only the K-vertices (with no Steiner points), such that each vertex in $K - S_i$ has degree $O(\log n)$ in $T(K)$.

[4] An affected component does not necessarily have F-vertices in it, so it could remain as one intact part.

(5) (New) **Small neighbor-sets:** For every $K \in \mathcal{K}(S_i)$, it holds that $|N(\mathcal{H}_K(S_i))| = O(f \log n)$.

2.6 Putting It All Together

We can now give a rough description of how the labels are constructed and used to answer queries, ignoring some nuances and technicalities.

Constructing Labels. We focus on the label of an (assumed to be) faulty vertex x, as these do most of the work during queries. The label $L(x)$ is a concatenation of $f+1$ labels $L_i(x)$, one for each hierarchy $\mathcal{H}(S_i)$. We only care about sets S_i where $x \notin S_i$, as the S_i vertices are considered safe (otherwise, we leave $L_i(x)$ empty). We construct an auxiliary shortcut graph $\hat{G}(\mathcal{H}(S_i))$ based on $\mathcal{H}(S_i)$, by adding typed shortcut edges, exactly as explained in Section 2.4. Let K_x be the component containing x.

- For each neighbor y of x in $T(K_x)$, of which there are $O(\log n)$ since $x \notin S_i$, let $T_y(K_x)$ be the subtree rooted at y. We store $\mathsf{sketch}(V(T_y(K_x)))$, constructed with respect to $\hat{G}(\mathcal{H}(S_i))$. These are akin to the subtree-sketches from Section 2.2.
- Next, we refer to the $O(\log n)$ components $K \supseteq K_x$, which we know will be affected.
  - Our main concern is the ability to delete edges with type "K" from the sketches, since these are unrelible when K is affected. We thus store $\mathsf{sketch}(\hat{E}_K(v))$, the sketch of the "$K$"-type edges touching v, for every $v \in N(\mathcal{H}_K(S_i))$.
  - Also, to account for the possibility that no F-vertex will lend in K, so K will be a part in the initial Borůvka partition, we store $\mathsf{sketch}(K)$

The length of labels is bounded as follows. First, the sketches are constructed using the fault-tolerant sampling approach of Section 2.2, hence a single $\mathsf{sketch}(\cdot)$ takes up $\tilde{O}(f^2)$ bits. As neighbor-sets $N(\mathcal{H}_K(S_i))$ are of size $\tilde{O}(f)$, the label $L_i(x)$ consists of $\tilde{O}(f^3)$ bits. The final label $L(x)$, which concatenates $f+1$ different $L_i(x)$-labels, thus consists of $\tilde{O}(f^4)$ bits. In fact, we can reduce one f-factor from the size of sketches by using an "orientation trick", explained in the following Section 2.7.

Answering Queries. Fix a query $\langle s, t, F\rangle$ with $F \cap S_i \neq \emptyset$. It defines the affected components in $\mathcal{K}(S_i)$, and hence the query graph G^* as in Section 2.4, which is the subgraph of $\hat{G}(\mathcal{H}(S_i))$ induced on vertices in affected components, but only with the edges of unaffected types. The parts to which each tree $T(K)$ of an affected component K breaks after the failure of F constitute the initial partition for running Borůvka in $G^* - F$. We compute part-sketches by XORing subtree sketches, similarly to Section 2.2. We also delete the bad edges from these, using the stored $\mathsf{sketch}(\hat{E}_K(v))$ of every affected K and $v \in N(\mathcal{H}_K(S_i))$, so that the part-sketches now represent G^*. Using these sketches we can simulate the Borůvka algorithm in $G^* - F$, and check if the initial parts containing s, t ended up in the same final part. We answer that s, t are connected in $G - F$ if and only if this is the case.

2.7 Improvement: The "Orientation Trick"

In fact, we can save one f factor in the length of the labels described in the previous section, by an idea we refer to as the "orientation trick". We first explain how this trick can be applied for the intuitive approach of Section 2.2, where we are given a low-degree spanning tree T of G.

Apply Nagamochi-Ibaraki [23] sparsification, and replace G with an *f-vertex connectivity certificate*: a subgraph where all connectivity queries under $\leq f$ vertex faults have the same answers as in G. The certificate (which we assume is G itself from now on) has *arboricity* $\leq f$, meaning we can *orient* the edges of G so that vertices have outdegrees at most f. We do not think of the orientation as making G directed; an edge $\{u, v\}$ oriented as $u \to v$ is still allowed to be traversed from v to u. Rather, the orientation is a trick that lets us mix the two strategies we have for avoiding edges incident to F when extracting outgoing edges from sketches: explicit deletion (as in Dory-Parter, Section 2.1), or fault-tolerant sketching (as in Section 2.2).

The idea works roughly as follows. We generate just f random subgraph $G_1, \ldots, G_f$ (instead of f^2). Each G_i is generated by sampling each vertex w.p. $1/f$, and only keeping the edges oriented as $u \to v$ with v sampled (even if u is not sampled). We now have f basic sketches instead of every G-sketch; one for each G_i. When we extract an outgoing edge from a part P, we can avoid edges oriented as $P \to F$, i.e., outgoing edges from P that are *incoming* to a failed vertex from F. However, we may still get edges oriented in the reverse $F \to P$ direction. It therefore remains to delete from the sketches the edges that are outgoing from F-vertices. To this end, we first replace the independent sampling in generating the sketches with *pairwise independent hash functions*, maintaining the ability to extract an outgoing edge with constant probability. Now, each failed vertex can store all its $\leq f$ outgoing edges along with a short $\tilde{O}(1)$ random seed, from which we can deduce their sketches for explicit deletion. So now, each failed vertex stores only f basic sketches for each incident tree edge, and additional $\tilde{O}(f)$ information regarding its *outgoing* edges, resulting in $\tilde{O}(f\Delta(T))$-bit labels.

In order to apply this trick on the hierarchy-based sketches, i.e., upon each auxiliary shortcut graph $\hat{G}(\mathcal{H}(S_i))$ constructed for hierarchy $\mathcal{H}(S_i)$, we develop a different sparsification procedure than [23], which is sensitive to the different types of edges, and produces a low-arboricity "certificate" that can replace $\hat{G}(S_i)$.

2.8 Organization

In Section 3 we construct the new low-degree hierarchies. Section 4 defines the auxiliary graphs used in the preprocessing and query stages, and walks through their use in the query algorithm at a high level. In Section 5 we construct the $\tilde{O}(f^3)$-bit vertex labels, and Section 6 gives the implementation details of the query algorithm. Proofs and figures are omitted due to space constraints; they appear in the full version [27].

3 A NEW LOW-DEGREE DECOMPOSITION THEOREM

In this section, we construct the new low-degree hierarchies on which our labeling scheme is based. Recall our starting point is Duan and Pettie's low degree hierarchy [12], whose properties are overviewed in Section 2.3. We state them succinctly and formally in the following Theorem 3.1.

Theorem 3.1 (Modification of [12, Section 4]). *There is a partition $\mathcal{C}$ of $V(G)$ and a rooted hierarchy tree $\mathcal{H}^0 = (\mathcal{C}, E(\mathcal{H}^0))$ with the following properties.*

1. *$\mathcal{H}^0$ has height at most $\log n$.*
2. *For $\gamma, \gamma' \in \mathcal{C}$, $\gamma \prec \gamma'$ denotes that γ is a strict descendant of γ'. If $\{u, v\} \in E(G)$, and $\gamma_u, \gamma_v \in \mathcal{C}$ are the parts containing u, v, then $\gamma_u \prec \gamma_v$ or $\gamma_v \preceq \gamma_u$.*
3. *For every $\gamma \in \mathcal{C}$, the graph induced by $V(\mathcal{H}^0_\gamma) \stackrel{\text{def}}{=} \bigcup_{\gamma' \preceq \gamma} \gamma'$ is connected. In particular, for every $\gamma \in \mathcal{C}$ and child γ', $E \cap (\gamma \times V(\mathcal{H}^0_{\gamma'})) \neq \emptyset$.*
4. *Each $\gamma \in \mathcal{C}$ is spanned by Steiner tree $T^0(\gamma)$ (that may have Steiner vertices not in γ) with maximum degree at most 4. Further, for $T^\cup \stackrel{\text{def}}{=} \bigcup_{\gamma \in \mathcal{C}} T^0(\gamma)$, it holds that the maximum degree in $T^\cup$ is at most $2 \log n$.*

We note that Long and Saranurak [22] recently gave a fast construction of a low degree hierarchy, in $O(m^{1+o(1)})$ time, while increasing the degree bound of Theorem 3.1(4) from 4 to $n^{o(1)}$. The space for our labeling scheme depends linearly on this degree bound, so we prefer Theorem 3.1 over [22] even though the construction time is higher.

The following Theorem 3.2 states the properties of the new low-degree hierarchies constructed in this paper, as overviewed in Section 2.5.

Theorem 3.2 (New Low-Degree Hierarchies). *Let $f \geq 1$ be an integer. There exists a partition $(S_1, \ldots, S_{f+1})$ of $V(G)$, such that each $S \in \{S_1, \ldots, S_{f+1}\}$ is associated with a hierarchy $\mathcal{H}(S)$ of components $\mathcal{K}(S)$ that partition $V(G)$, and the following hold.*

1. *$\mathcal{H} = \mathcal{H}(S) = (\mathcal{K}(S), E(\mathcal{H}(S)))$ is a* coarsening *of $\mathcal{H}^0$. $\mathcal{K}(S)$ is obtained by unifying connected subtrees of $\mathcal{H}^0$. $\mathcal{H}$ inherits Properties 1–3 of Theorem 3.1. In particular, define $\mathcal{H}_K$ to be the subhierarchy rooted at $K \in \mathcal{K}(S)$, and $V(\mathcal{H}_K) \stackrel{\text{def}}{=} \bigcup_{K' \preceq K} K'$. Then the graph induced by $V(\mathcal{H}_K)$ is connected, and if K' is a child of K then $E \cap (K \times V(\mathcal{H}_{K'})) \neq \emptyset$.*
2. *Each $K \in \mathcal{K}(S)$ has a spanning tree $T(K)$ in the subgraph of G induced by K. All vertices in $K - S$ have degree at most $3 \log n$ in $T(K)$, whereas S-vertices can have arbitrarily large degree.*
3. *For $K \in \mathcal{K}(S)$, define $N(\mathcal{H}_K)$ to be the set of vertices in $V - V(\mathcal{H}_K)$ that are adjacent to some vertex in $V(\mathcal{H}_K)$. Then $|N(\mathcal{H}_K)| = O(f \log n)$.*

The remainder of this section constitutes a proof of Theorem 3.2. We first choose the partition $(S_1, \ldots, S_f)$ of V uniformly at random among all partitions of V into $f + 1$ sets.

Fix some $S = S_i$. We now explain how its corresponding hierarchy $\mathcal{H} = \mathcal{H}(S)$ is constructed. We obtain $\mathcal{K} = \mathcal{K}(S)$ from $\mathcal{C}$ by iteratively unifying connected subtrees of the component tree $\mathcal{H}^0$. Initially $\mathcal{K} = \mathcal{C}$ and $\mathcal{H} = \mathcal{H}^0$. By Theorem 3.1 each $K = \gamma_i \in \mathcal{C}_i$ is initially spanned by a degree-4 Steiner tree $T(K) = T^0(\gamma_i)$ in $T_{i+1} - B_{i+1}$. We process each $\gamma \in \mathcal{C}$ in postorder (with respect to the tree $\mathcal{H}^0$). Suppose, in the current state of the partition, that $K_\gamma \in \mathcal{K}$ is the part containing γ. While there exists a $K_1 \in \mathcal{K}$ such that K_1 is a descendant of K_γ and one of the following criteria hold:

(i) $T^\cup \cap (K_1 \times \gamma) \neq \emptyset$, or

(ii) $E \cap (K_1 \times (\gamma \cap S)) \neq \emptyset$,

then we will *unify* a connected subtree of $\mathcal{H}$ that includes K_γ, K_1 and potentially many other parts of the current partition $\mathcal{K}$. Let e_γ be an edge from set (i) or (ii). If K_1 is a child of K_γ then we simply replace K_γ, K_1 in $\mathcal{K}$ with $K_\gamma \cup K_1$, spanned by $T(K_\gamma) \cup \{e_\gamma\} \cup T(K_1)$. In general, let K_0 be the child of K_γ that is ancestral to K_1. We call a procedure $\mathsf{Unify}(K_0, \{K_1\})$ that outputs a set of edges E' that connects K_0, K_1 and possibly other components. We then replace the components in $\mathcal{K}$ spanned by $E' \cup \{e_\gamma\}$ with their union, whose spanning tree consists of the constituent spanning trees and $E' \cup \{e_\gamma\}$. This unification process is repeated so long as there is *some* γ, *some* descendant K_1, and *some* edge e_γ in sets (i) or (ii).

In general Unify takes two arguments: a K_0 and a set $\mathcal{L}$ of descendants of K_0.

Algorithm 1 $\mathsf{Unify}(K_0, \mathcal{L})$

Input: A root component K_0 and set $\mathcal{L}$ of descendants of K_0.
Output: A set of edges E' joining $\{K_0\} \cup \mathcal{L}$ (and possibly others) into a single tree.

```
if {K_0} ∪ L = {K_0} then
    return ∅                                   ▷ Nothing to do
E' ← ∅
Define K_0^1, ..., K_0^t to be the children of K_0 that are ancestral to some component in L.
for i = 1 to t do
    Let L_i ⊆ L be the descendants of K_0^i.
    Let e_i ∈ E ∩ (K_0 × V(H_{K_0^i})) be an edge joining K_0 and some descendant K_i of K_0^i.
    E_i ← Unify(K_0^i, L_i ∪ {K_i}).
    E' ← E' ∪ E_i ∪ {e_i}
return E'
```

Lemma 3.3. $\mathsf{Unify}(K_0, \mathcal{L})$ *returns an edge set $E' \subseteq E(G)$ that forms a tree on a subset U of the components in the current state of the hierarchy $\mathcal{H}$. The subgraph of $\mathcal{H}$ induced by U is a connected subtree rooted at K_0 and containing $\{K_0\} \cup \mathcal{L}$.*

Let $\mathcal{H}(S) = (\mathcal{K}(S), E(\mathcal{H}(S)))$ be the coarsened hierarchy after all unification events, and $T(K)$ be the spanning tree of $K \in \mathcal{K}(S)$. Lemma 3.3 guarantees that each unification event is on a connected subtree of the current hierarchy. Thus, the final hierarchy $\mathcal{H}(S)$ satisfies Part 1 of Theorem 3.2.

Lemma 3.4. *$T(K)$ is a spanning tree of K; it contains no Steiner vertices outside of K. For all $v \in K - S$, $\deg_{T(K)}(v) \leq 3 \log n$.*

Part 2 of Theorem 3.2 follows from Lemma 3.4. Only Part 3 depends on how we choose the partition $(S_1, \ldots, S_{f+1})$. Recall this partition was selected uniformly at random, i.e., we pick a coloring function $\phi : V \to \{1, \ldots, f+1\}$ uniformly at random and let $S_i = \{v \in V \mid \phi(v) = i\}$. Consider any $\gamma' \in \mathcal{C}$ with $|N(\mathcal{H}^0_{\gamma'})| \geq 3(f+1) \ln n$. For any such γ' and any index $i \in \{1, \ldots, f+1\}$,

$$\Pr[N(\mathcal{H}^0_{\gamma'}) \cap S_i = \emptyset] \leq \left(1 - \frac{1}{f+1}\right)^{3(f+1)\ln n} < n^{-3}.$$

Taking a union bound over all (γ', i), $N(\mathcal{H}^0_{\gamma'}) \cap S_i \neq \emptyset$ with probability at least $1 - 1/n$. Assuming this holds, let e_γ be an edge joining

an S_i-vertex in γ and some vertex in $\gamma' \prec \gamma$. When processing γ, we would therefore find the type-(ii) edge e_γ that triggers the unification of γ, γ'. Thus, γ' cannot be the root-component of any $K \in \mathcal{K}(S_i)$ in the final hierarchy $\mathcal{H}(S_i)$, for any $i \in \{1, \ldots, f+1\}$.

This concludes the proof of Theorem 3.2.

4 AUXILIARY GRAPH STRUCTURES

In this section, we define and analyze the properties of several auxiliary graph structures, that are based on the low-degree hierarchy $\mathcal{H}(S)$ of Theorem 3.2 and on the query $\langle s, t, F\rangle$, as described at a high-level in Section 2.4.

Recall that $S \in \{S_1, \ldots, S_{f+1}\}$ are vertices that are not allowed to fail, so whenever the query $\langle s, t, F\rangle$ is known, $S = S_i$ refers to a part for which $S_i \cap F = \emptyset$.

We continue to use the notation $\mathcal{H} = \mathcal{H}(S)$, $\mathcal{K} = \mathcal{K}(S)$, $\mathcal{H}_K$, $V(\mathcal{H}_K)$, $N(\mathcal{H}_K)$, $T(K)$, etc. In addition, for $v \in V(G)$, $K_v \in \mathcal{K}$ is the component containing v. Also, for $u \in K \in \mathcal{K}$, $T_u(K)$ is the subtree of $T(K)$ rooted at u, where $T(K)$ is rooted arbitrarily at some vertex $r_K \in K$.

4.1 The "Shortcuts-Graph" $\hat{G}$ for $\mathcal{H}(S)$

We define $\hat{G} = \hat{G}(\mathcal{H}(S))$ as the *edge-typed* multi-graph, on the vertex set $V(G)$, constructed as follows: Start with G, and give its edges type *original*. For every component $K \in \mathcal{K}(S)$, add a clique on the vertex set $N(\mathcal{H}_K)$, whose edges have type "K." Intuitively, these are "shortcut edges" which represent the fact that any two vertices in $N(\mathcal{H}_K)$ are connected by a path in G whose internal vertices are all from $V(\mathcal{H}_K)$. We denote the set of $\hat{G}$-edges by $\hat{E} = \hat{E}(\mathcal{H}(S))$.

Lemma 4.1. *Let $e = \{u, v\} \in \hat{E}$.[5] Then K_u and K_v are related by the ancestry relation in $\mathcal{H}$.*

Lemma 4.2. *For any $K \in \mathcal{K}$, let $\hat{N}(\mathcal{H}_K)$ be all vertices in $V - V(\mathcal{H}_K)$ that are adjacent, in $\hat{G}$, to some vertex in $V(\mathcal{H}_K)$. Then $\hat{N}(\mathcal{H}_K) = N(\mathcal{H}_K)$.*

4.2 The Query Graph G^*

We now define and analyze notions that are based on the connectivity query $\langle s, t, F\rangle$ to be answered. A component $K \in \mathcal{K}(S)$ is *affected* by the query $\langle s, t, F\rangle$ if $V(\mathcal{H}_K) \cap (F \cup \{s, t\}) \neq \emptyset$. Note that if K is affected, then so are all its ancestor components. An edge $e = \{u, v\} \in \hat{E}$ of type χ is *valid* with respect to the query $\langle s, t, F\rangle$ if both the following hold:

(C1) $\chi =$ *original* or $\chi = K$ for some unaffected K, and
(C2) K_u and K_v are affected.

We denote the set of valid edges by $E^* = E^*(\mathcal{H}(S), \langle s, t, F\rangle)$. Intuitively, two vertices u, v in affected components are connected by a valid edge if there is a *reliable* path between u, v in G, whose internal vertices all lie in unaffected components, and therefore cannot intersect F. The *query graph* $G^* = G^*(\mathcal{H}(S), \langle s, t, F\rangle)$ is the subgraph of $\hat{G}$ consisting of all vertices lying in affected components of $\mathcal{H}(S)$, and all valid edges E^* w.r.t. the query $\langle s, t, F\rangle$.

[5]Throughout, we slightly abuse notation and write $e = \{u, v\}$ to say that e has endpoints u, v, even though there might be several different edges with these same endpoints, but with different types.

The following lemma gives the crucial property of G^* which we use to answer queries: To decide if s, t are connected in $G - F$, it suffices to determine their connectivity in $G^* - F$.

Lemma 4.3. *Let G^* be the query graph for $\langle s, t, F\rangle$ and hierarchy $\mathcal{H}(S)$. If $x, y \in V - F$ are vertices in affected components of $\mathcal{H}(S)$, then x and y are connected in $G - F$ iff they are connected in $G^* - F$.*

4.3 Strategy for Connectivity Queries

The query $\langle s, t, F\rangle$ determines the set $S = S_i$ for which $S \cap F = \emptyset$. The query algorithm deals only with $\mathcal{H}(S)$ and the graph $G^* = G^*(\mathcal{H}(S), \langle s, t, F\rangle)$. In this section we describe how the query algorithm works at a high level, in order to highlight what information must be stored in the vertex labels of s, t, F, and which operations must be supported by those labels.

The query algorithm depends on a sketch (probabilistic data structure) for handling a certain type of *cut query* [3, 18]. In subsequent sections we show that such a data structure exists, and can be encoded in the labels of the failed vertices. For the time being, suppose that for any vertex set $P \subseteq V(G^*)$, $\mathsf{sketch}(P)$ is some data structure subject to the operations

$\mathsf{Merge}(\mathsf{sketch}(P), \mathsf{sketch}(P'))$ **:** Returns $\mathsf{sketch}(P \oplus P')$.
$\mathsf{GetEdge}(\mathsf{sketch}(P), F)$ **:** If $F \cap P = \emptyset$, $|F| \leq f$, returns an edge $e \in E^* \cap (P \times (V - (P \cup F)))$ with probability $\delta = \Omega(1)$ (if any such edge exists), and FAIL otherwise.

It follows from Theorem 3.2 and $S \cap F = \emptyset$ that the graph $\bigcup_{\text{affected } K}(T(K) - F)$ consists of $O(f \log n)$ disjoint trees, whose union covers $V(G^*) - F$. Let $\mathcal{P}_0$ be the corresponding vertex partition of $V(G^*) - F$. Following [3, 10, 12, 18], we use Merge and GetEdge queries to implement an unweighted version of Borůvka's minimum spanning tree algorithm on $G^* - F$, in $O(\log n)$ parallel rounds. At round i we have a partition $\mathcal{P}_i$ of $V(G^*) - F$ such that each part of $\mathcal{P}_i$ is spanned by a tree in $G^* - F$, as well as $\mathsf{sketch}(P)$ for every $P \in \mathcal{P}_i$. For each $P \in \mathcal{P}_i$, we call $\mathsf{GetEdge}(\mathsf{sketch}(P), F)$, which returns an edge to another part of $\mathcal{P}_i$ with probability δ, since all edges to F are excluded. The partition $\mathcal{P}_{i+1}$ is obtained by unifying all parts of $\mathcal{P}_i$ joined by an edge returned by GetEdge; the sketches for $\mathcal{P}_{i+1}$ are obtained by calling Merge on the constituent sketches of $\mathcal{P}_i$. (Observe that distinct $P, P' \in \mathcal{P}_i$ are disjoint so $P \oplus P' = P \cup P'$.)

Once the final partition $\mathcal{P}_{O(\log n)}$ is obtained, we report *connected* if s, t are in the same part and *disconnected* otherwise. By Lemma 4.3, s, t are connected in $G - F$ iff they are connected in $G^* - F$, so it suffices to prove that $\mathcal{P}_{O(\log n)}$ is the partition of $G^* - F$ into connected components, with high probability.

Analysis. Let N_i be the number of parts of $\mathcal{P}_i$ that are not already connected components of $G^* - F$. We claim $\mathbb{E}[N_{i+1} | N_i] \leq N_i - (\delta/2)N_i$. In expectation, δN_i of the calls to GetEdge return an edge. If there are z successful calls to GetEdge, the z edges form a pseudoforest[6] and any pseudoforest on z edges has at most $\lceil z/2 \rceil$ connected components. Thus after $c \ln n$ rounds of Borůvka's algorithm, $\mathbb{E}[N_{c \ln n}] \leq n(1 - (\delta/2))^{c \ln n} < n^{1 - c\delta/2}$. By Markov's inequality, $\Pr[N_{c \ln n} \geq 1] \leq n^{1 - c\delta/2}$. In other words, when $c = \Omega(1/\delta)$, with high probability $N_{c \ln n} = 0$ and $\mathcal{P}_{c \ln n}$ is exactly the partition of $G^* - F$ into connected components. Thus,

[6]A subgraph that can be oriented so that all vertices have out-degree at most 1.

any connectivity query $\langle s, t, F\rangle$ is answered correctly, with high probability.

4.4 Classification of Edges

The graph G^* depends on $\mathcal{H}(S)$ and on the *entire query* $\langle s, t, F\rangle$. In contrast, the label of a query vertex x is constructed without knowing the rest of the query, so storing G^*-related information is challenging. However, we *do* know that all the ancestor-components of K_x are affected. This is the intuitive motivation for this section, where we express G^*-information in terms of individual vertices and (affected) components. Specifically, we provide technical structural lemmas that express cut-sets in G^* in terms of several simpler edge-sets, exploiting the structure of the hierarchy $\mathcal{H}(S)$. Sketches of the latter sets can be divided across the labels of the query vertices, which helps us to keep them succinct, as explained in the later Section 5, while enabling the Borůvka initialization described in Section 6.

Fix the hierarchy $\mathcal{H} = \mathcal{H}(S)$ and the query $\langle s, t, F\rangle$. Note that $\mathcal{H}$ determines the graph $\hat{G}$ whereas $\langle s, t, F\rangle$ further determines G^*. We define the following edge sets, where $K \in \mathcal{K}(S)$, $v \in V$

- $\hat{E}(v, K)$: the set of all $\hat{G}$-edges with v as one endpoint, and the other endpoint in K.
- $\hat{E}_K(v)$: the set of all $\hat{G}$-edges of type K incident to v.
- $\hat{E}_{\text{up}}(v) \stackrel{\text{def}}{=} \bigcup_{K \geq K_v} \hat{E}(v, K)$. I.e., the $\hat{G}$-edges incident to v having their other endpoint in an ancestor component of K_v, including K_v itself.
- $\hat{E}_{\text{down}}(v) \stackrel{\text{def}}{=} \bigcup_{K < K_v,\ K \text{ affected}} \hat{E}(v, K)$. I.e., the $\hat{G}$-edges incident to v having their other endpoint in an *affected* component which is a strict descendant of K_v.
- $\hat{E}_{\text{bad}}(v) \stackrel{\text{def}}{=} \bigcup_{K \text{ affected}} E_K(v)$. I.e., the $\hat{G}$-edges incident to v having affected types.
- $E^*(v)$: the set of all E^*-edges (valid $\hat{G}$-edges) incident to v, defined only when $v \in V(G^*)$.

We emphasize that despite their similarity, the notations $\hat{E}(v, K)$ and $\hat{E}_K(v)$ have entirely different meanings; in the first K serves as the *hosting component* of the non-v endpoints of the edges, while in the second, K is the *type* of the edges. Also, note that the first three sets only depend on the hierarchy $\mathcal{H} = \mathcal{H}(S)$ while the rest also depend on the query $\langle s, t, F\rangle$.

The following lemma expresses $E^*(v)$ in terms of $\hat{E}(v, K)$ and $\hat{E}_K(v)$ for affected $K \in \mathcal{K}(S)$.

Lemma 4.4. *Let v be a vertex in G^*. Then:*

$$\hat{E}_{\text{down}}(v) = \bigoplus_{K \text{ affected},\ v \in N(\mathcal{H}_K)} \hat{E}(v, K)\ . \tag{1}$$

$$\hat{E}_{\text{bad}}(v) = \bigoplus_{K \text{ affected},\ v \in N(\mathcal{H}_K)} \hat{E}_K(v)\ . \tag{2}$$

$$E^*(v) = \hat{E}_{\text{up}}(v) \oplus \hat{E}_{\text{down}}(v) \oplus \hat{E}_{\text{bad}}(v)\ . \tag{3}$$

We next consider cut-sets in G^*. For $U \subseteq V(G^*)$, let $E^*_{\text{cut}}(U)$ be the set of edges crossing the cut $(U, V(G^*) - U)$ in G^*.

Observation 4.5. $E^*_{\text{cut}}(U) = \bigoplus_{v \in U} E^*(v)$.

We end the section with the following Lemma 4.6 that provides a useful formula for cut-sets in G^*. The proof is by easy applications of Lemma 4.4 and Observation 4.5.

Lemma 4.6. *Let $U \subseteq V(G^*)$. Then*

$$E^*_{\text{cut}}(U) = \left(\bigoplus_{v \in U} E_{\text{up}}(v)\right) \oplus \left(\bigoplus_{K \text{ affected}} \bigoplus_{v \in U \cap N(\mathcal{H}_K)} \hat{E}(v, K) \oplus \hat{E}_K(v)\right). \tag{4}$$

4.5 Sparsifying and Orienting $\hat{G}$

In this section we set the stage for using the "orientation trick", overviewed in Section 2.7, that ultimately enables us to reduce the label size in our construction further. We show that we can effectively sparsify $\hat{G}$ to have arboricity $\tilde{O}(f^2)$, or equivalently, to admit an $\tilde{O}(f^2)$-outdegree orientation, while preserving, with high probability, the key property of G^* stated in Lemma 4.3, that x, y are connected in $G - F$ iff they are connected in $G^* - F$. This is formalized in the following lemma:

Lemma 4.7. *There is a randomized procedure that given the graph $\hat{G} = \hat{G}(\mathcal{H}(S))$, outputs a subgraph $\tilde{G}$ of $\hat{G}$ with the following properties.*

1. *$\tilde{G}$ has arboricity $O(f^2 \log^2 n)$. Equivalently, its edges can be oriented so that each vertex has outdegree $O(f^2 \log^2 n)$.*
2. *Fix any query $\langle s, t, F\rangle$, $|F| \leq f$, and consider the query graph $G^* = G^*(\mathcal{H}(S), \langle s, t, F\rangle)$. Let $\tilde{G}^* = G^* \cap \tilde{G}$ be the subgraph of G^* whose edges are present in $\tilde{G}$. Let $x, y \in V - F$ be two vertices in affected components. With high probability, x, y are connected in $G - F$ iff they are connected in $\tilde{G}^* - F$.*

Henceforth, we use $\hat{G}$ to refer to the sparsified and *oriented* version of $\hat{G}$ returned by Lemma 4.7, i.e., $\hat{G}$ is now $\tilde{G}$. Note that the edges of $\hat{G}$ now have two extra attributes: a *type* and an *orientation*. An oriented graph is not the same as a *directed* graph. When $\{u, v\}$ is oriented as $u \to v$, a path may still use it in either direction. Informally, the orientation serves as a tool to reduce the label size while still allowing GetEdge from Section 4.3 to be implemented efficiently. Each vertex u will store explicit information about its $\tilde{O}(f^2)$ incident out-edges that are oriented as $u \to v$.

5 SKETCHING AND LABELING

In this section, we first develop the specialized sketching tools that work together with each hierarchy $\mathcal{H} = \mathcal{H}(S)$, and then define the labels assigned by our scheme, which store such sketches.

5.1 Sketching Tools

Fix $S \in \{S_1, \ldots, S_{f+1}\}$ and the hierarchy $\mathcal{H} = \mathcal{H}(S)$ from Theorem 3.2. All presented definitions are with respect to this hierarchy $\mathcal{H}$.

IDs and Ancestry Labels. Before formally defining the sketches, we need several preliminary notions of identifiers and ancestry labels. We give each $v \in V$ a unique $\text{id}(v) \in [1, n]$, and also each component $K \in \mathcal{K}(S)$ has a unique $\text{id}(K) \in [1, n]$. We also assign simple *ancestry* labels:

Lemma 5.1. *One can give each $v \in V$ an $O(\log n)$-bit* ancestry label $\text{anc}(v)$. *Given* $\text{anc}(u)$ *and* $\text{anc}(v)$ *for $u, v \in V$, one can determine if $K_u \geq K_v$ and if equality holds. In the latter case, one can also determine if u is an ancestor of v in $T(K_u) = T(K_v)$ and if $u =$*

v. Ancestry labels are extended to components $K \in \mathcal{K}$, by letting $\mathsf{anc}(K) \stackrel{\text{def}}{=} \mathsf{anc}(r_K)$ *where r_K is the root of $T(K)$.*

The type of each $e \in \hat{E}$ is denoted by $\mathsf{type}(e) \in \{\perp\} \cup \{\mathsf{id}(K) \mid K \in \mathcal{K}\}$, where $\perp \notin [1,n]$ is a non-zero $O(\log n)$-bit string representing the *original* type. Let $\hat{E}_{\text{all}}$ be the set of all possible edges having two distinct endpoints from V and type from $\{\perp\} \cup [1,n]$. Each $e \in \hat{E}_{\text{all}}$ is defined by the ids of its endpoints, its type, and its orientation. Recall that $\hat{E}$-edges were oriented in Section 4.5; the orientation of $\hat{E}_{\text{all}} - \hat{E}$ is arbitrary. Let $\omega \stackrel{\text{def}}{=} \lceil \log(|\hat{E}_{\text{all}}|) \rceil = O(\log n)$.

The following lemma introduces *unique edge identifiers* (uids). It is a straightforward modification of [14, Lemma 2.3], which is based on the notion of ϵ-*biased sets* [24]:

Lemma 5.2 ([14]). *Using a random seed $\mathcal{S}_{\mathsf{id}}$ of $O(\log^2 n)$ bits, one can compute $O(\log n)$-bit identifiers $\mathsf{uid}(e)$ for each possible edge $e \in \hat{E}_{\text{all}}$, with the following properties:*

1. *If $E' \subseteq \hat{E}_{\text{all}}$ with $|E'| \neq 1$, then w.h.p. $\bigoplus_{e' \in E'} \mathsf{uid}(e') \neq \mathsf{uid}(e)$, for every $e \in \hat{E}_{\text{all}}$. That is, the bitwise-XOR of more than one* uid *is, w.h.p., not a valid* uid *of any edge.*[7]
2. *Let $e = \{u, v\} \in \hat{E}_{\text{all}}$. Then given $\mathsf{id}(u)$, $\mathsf{id}(v)$, $\mathsf{type}(e)$ and $\mathcal{S}_{\mathsf{id}}$, one can compute $\mathsf{uid}(e)$.*

Next, we define $O(\log n)$-bit *extended edge identifiers* (eids). For an edge $e = \{u, v\} \in \hat{E}_{\text{all}}$ oriented as $u \to v$,

$$\mathsf{eid}(e) \stackrel{\text{def}}{=} \langle \mathsf{uid}(e), \mathsf{id}(u), \mathsf{id}(v), \mathsf{type}(e), \mathsf{anc}(u), \mathsf{anc}(v) \rangle .$$

The point of using uids is the following Lemma 5.3, allowing to distinguish between eids of edges and "garbage strings" formed by XORing many of these eids:

Lemma 5.3. *Fix any $E' \subseteq \hat{E}$. Given $\bigoplus_{e \in E'} \mathsf{eid}(e)$ and the seed $\mathcal{S}_{\mathsf{id}}$, one can determine whether $|E'| = 1$, w.h.p., and therefore obtain $\mathsf{eid}(e)$ for the unique edge e such that $E' = \{e\}$.*

Defining Sketches. First, we take two pairwise independent hash families: a family Φ for hashing edges of functions $\varphi : \hat{E}_{\text{all}} \to [0, 2^\omega)$, and a family $\mathscr{H}$ for hashing vertices of functions $h : V \to [1, 2f]$. These serve to replace the independent sampling of edges and vertices in forming sketches, as described in Section 2.1 and Section 2.2 respectively, so as to enable the use of the "orientation trick" overviewed in Section 2.7.

Let $p \stackrel{\text{def}}{=} \lceil c \log n \rceil$ for a sufficiently large constant c. For any $q \in [1,p]$ and $i \in [1,f]$ we choose random hash functions $h_{q,i} \in \mathcal{H}$ and $\varphi_{q,i} \in \Phi$. Recalling that $\hat{G}$ is *oriented*, the subgraph $\hat{G}_{q,i}$ of $\hat{G}$ has the same vertex set V, and its edge set is defined by

$$E(\hat{G}_{q,i}) \stackrel{\text{def}}{=} \{e = \{u,v\} \in E(\hat{G}) \mid \text{orientation is } u \to v \text{ and } h_{q,i}(v) = 1\}.$$

We then create the corresponding nested family of edge-subsets for $\hat{G}_{q,i}$, defined as

$$E(\hat{G}_{q,i}) = \hat{E}_{q,i,0} \supseteq \hat{E}_{q,i,1} \supseteq \cdots \supseteq \hat{E}_{q,i,\omega}$$

where

$$\hat{E}_{q,i,j} \stackrel{\text{def}}{=} \{e \in E(\hat{G}_{q,i}) \mid \varphi_{q,i}(e) < 2^{\omega - j}\}.$$

[7] We emphasize that this holds for any *fixed* E' w.h.p., and not for all $E' \subseteq \hat{E}_{\text{all}}$ simultaneously.

Now, for an edge subset $E' \subseteq \hat{E}$, we define its sketch as follows. For $q \in [1,p]$, $i \in [1,f]$:

$$\mathsf{sketch}_{q,i}(E') \stackrel{\text{def}}{=} \left\langle \bigoplus_{e \in E' \cap \hat{E}_{q,i,0}} \mathsf{eid}(e), \ \ldots, \ \bigoplus_{e \in E' \cap \hat{E}_{q,i,\omega}} \mathsf{eid}(e) \right\rangle,$$

$$\mathsf{sketch}_{q}(E') \stackrel{\text{def}}{=} \left\langle \mathsf{sketch}_{q,1}(E'), \ldots, \mathsf{sketch}_{q,f}(E') \right\rangle,$$

$$\mathsf{sketch}(E') \stackrel{\text{def}}{=} \left\langle \mathsf{sketch}_{1}(E'), \ldots, \mathsf{sketch}_{p}(E') \right\rangle.$$

We can view sketch as a 3D array with dimensions $p \times f \times (\omega + 1)$, which occupies $O(pf\omega \cdot \log n) = O(f \log^3 n)$ bits.

Observation 5.4. *Sketches are* linear *w.r.t. the $\oplus$ operator: if $E_1, E_2 \subseteq \hat{E}$ then $\mathsf{sketch}_{q,i}(E_1 \oplus E_2) = \mathsf{sketch}_{q,i}(E_1) \oplus \mathsf{sketch}_{q,i}(E_2)$, and this property is inherited by* $\mathsf{sketch}(\cdot)$.

Note that hash functions in $\mathscr{H}, \Phi$ can be specified in $O(\log n)$ bits, so a random seed $\mathcal{S}_{\mathsf{hash}}$ of $O(f \log^2 n)$ bits specifies *all* hash functions $\{h_{q,i}, \varphi_{q,i}\}$. The following lemma essentially states we can use this small seed to compute sketches from eids:

Lemma 5.5. *Given the seed $\mathcal{S}_{\mathsf{hash}}$ and $\mathsf{eid}(e)$ of some $e \in \hat{E}$, one can compute the entire* $\mathsf{sketch}(\{e\})$.

The following Lemma 5.6 provides the key property of our sketches: an implementation of the GetEdge function (needed to implement connectivity queries, see Section 4.3), so long as the edge set *contains no edges oriented from an F-vertex.*

Lemma 5.6. *Fix any $F \subseteq V$, $|F| \leq f$, and let $E' \subseteq \hat{E}$ be a set of edges that contains no edges oriented from an F-vertex, and at least one edge with both endpoints in $V - F$. Then for any $q \in [1,p]$, with constant probability, some entry of $\mathsf{sketch}_q(E')$ is equal to $\mathsf{eid}(e)$, for some $e \in E'$ with both endpoints in $V - F$.*

We end this section by defining sketches *for vertex subsets*, as follows:

$$\mathsf{sketch}_{\mathsf{up}}(U) \stackrel{\text{def}}{=} \bigoplus_{u \in U} \mathsf{sketch}(\hat{E}_{\mathsf{up}}(u)) \qquad \text{for } U \subseteq V(G),$$

$$\mathsf{sketch}^*(U) \stackrel{\text{def}}{=} \bigoplus_{u \in U} \mathsf{sketch}(E^*(u)) \qquad \text{for } U \subseteq V(G^*).$$

Note that $\mathsf{sketch}_{\mathsf{up}}(U)$ depends only on the hierarchy $\mathcal{H} = \mathcal{H}(S)$, and thus it can be computed by the labeling algorithm. However, $\mathsf{sketch}^*(U)$ also depends on the query $\langle s, t, F \rangle$, so it is only possible to compute such a $\mathsf{sketch}^*(\cdot)$ at query time.

5.2 The Labels

We are now ready to construct the vertex labels. We first construct auxiliary labels $L_{\mathcal{H}(S)}(K)$ *for the components* of $\mathcal{H}(S)$ (Algorithm 2), then define the vertex labels $L_{\mathcal{H}(S)}(v)$ associated with $\mathcal{H}(S)$ (Algorithm 3). The final vertex label $L(v)$ is the concatenation of all $L_{\mathcal{H}(S_i)}(v)$, $i \in \{1, \ldots, f+1\}$.

The final labels. The final label $L(v)$ is the concatenation of the labels for each of the hierarchies $\mathcal{H}(S_1), \ldots, \mathcal{H}(S_{f+1})$ from Theorem 3.2.

$$L(v) \stackrel{\text{def}}{=} \left\langle L_{\mathcal{H}(S_1)}(v), L_{\mathcal{H}(S_2)}(v), \ldots, L_{\mathcal{H}(S_{f+1})}(v) \right\rangle.$$

Algorithm 2 Creating label $L_{\mathcal{H}(S)}(K)$ of a component $K \in \mathcal{K}(S)$

store $\mathsf{id}(K)$ and $\mathsf{anc}(K)$
store $\mathsf{sketch}_{\mathrm{up}}(K)$
for each $v \in N(\mathcal{H}_K)$ **do**
  store $\mathsf{id}(v)$ and $\mathsf{anc}(v)$
  store $\mathsf{sketch}(\hat{E}(v,K) \oplus \hat{E}_K(v))$

Algorithm 3 Creating label $L_{\mathcal{H}(S)}(v)$ of a vertex $v \in V$

store $\mathcal{S}_{\mathsf{id}}$ and $\mathcal{S}_{\mathsf{hash}}$
store $\mathsf{id}(v)$ and $\mathsf{anc}(v)$
for $K \geq K_v$ **do**
  store $L_{\mathcal{H}}(K)$
if $v \notin S$ **then**
  store $\mathsf{sketch}_{\mathrm{up}}(T_v(K_v))$
  for each child u of v in $T(K_v)$ **do**
    store $\mathsf{id}(u)$ and $\mathsf{anc}(u)$
    store $\mathsf{sketch}_{\mathrm{up}}(T_u(K_v))$
  for each edge $e = \{v,u\} \in \hat{E}$ incident to v, oriented as $v \to u$ **do**
    store $\mathsf{eid}(e)$

Length analysis. First, fix $\mathcal{H} = \mathcal{H}(S)$. The bit length of a component label $L_{\mathcal{H}(S)}(K)$ is dominated by $|N(\mathcal{H}_K)|$ times the length of a sketch$(\cdot)$. By Theorem 3.2(3), $|N(\mathcal{H}_K)| = O(f \log n)$, resulting in $O(f^2 \log^4 n)$ bits for $L_{\mathcal{H}(S)}(K)$. A vertex label $L_{\mathcal{H}(S)}(v)$ stores $L_{\mathcal{H}(S)}(K)$ for every $K \geq K_v$. There are at most $\log n$ such components K by Theorem 3.2(1), resulting in $O(f^2 \log^5 n)$ bits for storing the component labels. In case $v \notin S$, the bit length of the $\mathsf{sketch}_{\mathrm{up}}(\cdot)$ information stored is dominated by $\deg_{T(K_v)}(v)$ times the length of a $\mathsf{sketch}_{\mathrm{up}}(\cdot)$. By Theorem 3.2(2), $\deg_{T(K_v)}(v) < 3 \log n$, so this requires additional $O(f \log^4 n)$ bits. The eids stored are of edges oriented *away* from v, and, by Lemma 4.7, there are $O(f^2 \log^2 n)$ such edges, so they require an additional $O(f^2 \log^3 n)$ bits. The length of $L_{\mathcal{H}(S)}(v)$ can therefore be bounded by $O(f^2 \log^5 n)$ bits. In total $L(v)$ has bit length $O(f^3 \log^5 n)$.

6 ANSWERING QUERIES

We now explain how the high-level query algorithm of Section 4.3 is implemented. Let $\langle s, t, F\rangle$ be the query. To implement the Borůvka steps, we need to initialize all sketches for the initial partition $\mathcal{P}_0$ in order to support GetEdge and Merge.

The query algorithm first identifies a set S_i for which $S_i \cap F = \emptyset$; we only use information stored in $L_{\mathcal{H}(S_i)}(v)$, for $v \in F \cup \{s,t\}$. Recall that the high-level algorithm consists of $p = \Theta(\log n)$ *rounds*. Each round $q \in \{1, \ldots, p\}$ is given as input a partition $\mathcal{P}_{q-1}$ of $V(G^*) - F$ into connected *parts*, i.e., such that the subgraph of G^* induced by each $P \in \mathcal{P}_{q-1}$ is connected. It outputs a coarser partition $\mathcal{P}_q$, obtained by merging parts in $\mathcal{P}_{q-1}$ that are connected by edges of $G^* - F$. Merge is easy to implement as all of our sketches are linear w.r.t. $\oplus$. Lemma 5.6 provides an implementation of GetEdge for a part P, given a sketch of the edge-set $E^*_{\mathrm{cut}}(P) - E^*(F \to P)$, where $E^*(F \to P) \subseteq E^*_{\mathrm{cut}}(P)$ is the set of E^*-edges oriented from F to P.

We need to maintain the following invariants at the end of round $q \in \{0, 1, \ldots, p\}$.

(I1) We have an *ancestry representation* $\{\mathsf{anc}(P) \mid P \in \mathcal{P}_q\}$ for the partition, so that given any $\mathsf{anc}(v)$, $v \in V(G^*)$, we can locate which part P contains v.

(I2) For each part $P \in \mathcal{P}_q$, we know $\mathsf{sketch}^*_F(P) \stackrel{\text{def}}{=} \mathsf{sketch}^*(P) \oplus \mathsf{sketch}(E^*(F \to P))$, which is the sketch of the edge set $E^*_{\mathrm{cut}}(P) \oplus E^*(F \to P) = E^*_{\mathrm{cut}}(P) - E^*(F \to P)$.

Initialization. For an affected component $K \in \mathcal{K}(S_i)$, let $\mathcal{T}_F(K)$ be the set of connected components of $T(K) - F$. The initial partition is

$$\mathcal{P}_0 = \bigcup_{K \text{ affected}} \mathcal{T}_F(K).$$

Each $Q \in \mathcal{T}_F(K)$ can be defined by a *rooting vertex* r_Q, that is either the root r_K of $T(K)$, or a $T(K)$-*child* of some $x \in F \cap K$, as well as a set of *ending faults* F_Q containing all $x \in F \cap T_{r_Q}(K)$ having no strict ancestors from F in $T_{r_Q}(K)$. It may be that $F_Q = \emptyset$. Then,

$$Q = T_{r_Q}(K) - \bigcup_{x \in F_Q} T_x(K) = T_{r_Q}(K) \oplus \bigoplus_{x \in F_Q} T_x(K)\,. \tag{5}$$

The last equality holds as F_Q contains mutually unrelated vertices in $T_{r_Q}(K)$. It is easily verified that the $\mathsf{anc}(\cdot)$-labels of all roots of affected components, faults, and children of faults, are stored in the given input labels. By Lemma 5.1, we can deduce the ancestry relations between all these vertices. It is then straightforward to find, for each $Q \in \mathcal{P}_0$, its ancestry representation given by $\mathsf{anc}(Q) \stackrel{\text{def}}{=} \big\langle \mathsf{anc}(r_Q), \{\mathsf{anc}(x) \mid x \in F_Q\} \big\rangle$. This clearly satisfies (I1) for $\mathcal{P}_0$.

We now turn to the computation of $\mathsf{sketch}^*_F(Q) = \mathsf{sketch}^*(Q) \oplus \mathsf{sketch}(E^*(F \to Q))$. Lemma 4.6 shows how to compute $\mathsf{sketch}^*(Q)$ for $Q \in \mathcal{P}_0$.

Lemma 6.1. *For any $Q \in \mathcal{T}_F(K)$,*

$$\mathsf{sketch}^*(Q) = \tag{6}$$
$$\mathsf{sketch}_{\mathrm{up}}(Q) \oplus \bigoplus_{K \text{ affected}} \bigoplus_{v \in N(\mathcal{H}_K) \cap Q} \mathsf{sketch}(\hat{E}(v,K) \oplus \hat{E}_K(v)).$$

It follows from $F \cap S_i = \emptyset$ that for each affected $K \in \mathcal{K}(S_i)$ and $Q \in \mathcal{T}_F(K)$, $\mathsf{sketch}_{\mathrm{up}}(T_a(K))$ for each $a \in \{r_Q\} \cup F_Q$ can be found in the input vertex labels. This lets us compute $\mathsf{sketch}_{\mathrm{up}}(Q)$ using Eqn. (5) and linearity. By Eqn. (6), computing $\mathsf{sketch}^*(Q)$ now amounts to finding $\mathsf{sketch}(\hat{E}(v,K) \oplus \hat{E}_K(v))$ for each affected K and $v \in N(\mathcal{H}_K) \cap Q$. The label $L_{\mathcal{H}(S_i)}(K)$ stores this sketch for each $v \in N(\mathcal{H}_K)$; moreover, we can check if $v \in Q$ using the ancestry labels $\mathsf{anc}(v), \mathsf{anc}(Q)$.

By definition $\mathsf{sketch}(E^*(F \to Q)) = \bigoplus_e \mathsf{sketch}(\{e\})$, where the $\bigoplus$-sum is over all $e = \{a,v\}$ oriented as $a \to v$, where $a \in F, v \in Q$ and $\mathsf{type}(e)$ is not an affected component. (If $\mathsf{type}(e)$ is affected, $e \notin E^*$.) By Lemma 5.5, each such $\mathsf{sketch}(\{e\})$ can be constructed from $\mathsf{eid}(e)$ stored in $L_{\mathcal{H}(S_i)}(a)$, and we can check whether $v \in Q$ using $\mathsf{anc}(v), \mathsf{anc}(Q)$. In this way we can construct $\mathsf{sketch}^*_F(Q)$ for each $Q \in \mathcal{P}_0$, satisfying (I2).

Executing round q. For each $P \in \mathcal{P}_{q-1}$, we use Lemma 5.6 applied to $\mathsf{sketch}^*_{F,q}(P)$ (i.e., the qth subsketch of $\mathsf{sketch}^*_F(P)$) to implement GetEdge: with constant probability it returns the $\mathsf{eid}(e_P)$ for a single cut edge $e_P = \{u,v\} \in E^*_{\mathrm{cut}}(P)$ with $u \in P, v \in (V(G^*) -$

$(P \cup F)$), or reports FAIL otherwise. By (I1), given $\text{anc}(v)$ we can locate which part $P' \in \mathcal{P}_{q-1}$ contains v. Note that since GetEdge only depends on $\text{sketch}^*_{F,q}(P)$, its FAIL-probability is independent of the outcome of rounds $1, \ldots, q-1$.

The output partition $\mathcal{P}_q$ of round q is obtained by merging the connected parts of $\mathcal{P}_{q-1}$ along the discovered edges $\{e_P\}$. This ensures the connectivity of each new part in $G^* - F$. The ancestry representation of a new part $R \in \mathcal{P}_q$ is $\text{anc}(R) = \{\text{anc}(P) \mid R \supseteq P \in \mathcal{P}_{q-1}\}$, which establishes (I1) after round q. We then compute

$$\begin{aligned}
&\bigoplus_{P \in \mathcal{P}_{q-1}: P \subseteq R} \text{sketch}^*_F(P) \\
&= \bigoplus_{P \in \mathcal{P}_{q-1}: P \subseteq R} (\text{sketch}^*(P) \oplus \text{sketch}(E^*(F \to P)) \\
&= \text{sketch}^*(R) \oplus \text{sketch}(E^*(F \to R)) = \text{sketch}^*_F(R). \qquad (7)
\end{aligned}$$

Eqn. (7) follows from linearity (Observation 5.4), disjointness of the parts $\{P \in \mathcal{P}_{q-1} \mid P \subseteq R\}$, and disjointness of the edge sets $\{E^*(F \to P) \mid P \subseteq R\}$. This establishes (I2) after round q.

Finalizing. After executing the final round p, we use the ancestry representations for $\mathcal{P}_p$ and $\text{anc}(s), \text{anc}(t)$ to find the parts $P_s, P_t \in \mathcal{P}_p$ with $s \in P_s$, $t \in P_t$. We output *connected* iff $P_s = P_t$.

With high probability, the implementation of GetEdge using Lemma 5.3 and Lemma 5.6 reports no false positives, i.e., an edge that is *not* in the cut-set. Assuming no false positives, the correctness of the algorithm was established in Section 4.3. It is straightforward to prepare the initial sketches for $P \in \mathcal{P}_0$ in time $\tilde{O}(f^4)$, which is dominated by enumerating the $\tilde{O}(f^3)$ edges $e \in \bigcup_{P \in \mathcal{P}_0} E^*(F \to P)$ and constructing $\text{sketch}(\{e\})$ in $\tilde{O}(f)$ time. The time to execute Borůvka's algorithm is linear in the total length of all $\mathcal{P}_0$-sketches, which is $\tilde{O}(f^2)$. This concludes the proof of Theorem 1.1.

REFERENCES

[1] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. 2012. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings 44th ACM Symposium on Theory of Computing (STOC)*. 1199–1218. https://doi.org/10.1145/2213977.2214084

[2] Ittai Abraham, Shiri Chechik, Cyril Gavoille, and David Peleg. 2016. Forbidden-Set Distance Labels for Graphs of Bounded Doubling Dimension. *ACM Trans. Algorithms* 12, 2 (2016), 22:1–22:17. https://doi.org/10.1145/2818694

[3] Kook J. Ahn, Supdipto Guha, and Andrew McGregor. 2012. Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 459–467.

[4] Giuseppe Di Battista and Roberto Tamassia. 1996. On-line maintenance of triconnected components with SPQR-Trees. *Algorithmica* 15 (1996), 302–318.

[5] Otakar Borůvka. 1926. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti* 3 (1926), 37–58. In Czech..

[6] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. 2012. f-Sensitivity Distance Oracles and Routing Schemes. *Algorithmica* 63, 4 (2012), 861–882. https://doi.org/10.1007/s00453-011-9543-0

[7] Julia Chuzhoy and Sanjeev Khanna. 2009. An $O(k^3 \log n)$-Approximation Algorithm for Vertex-Connectivity Survivable Network Design. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 437–441.

[8] Bruno Courcelle, Cyril Gavoille, Mamadou Moustapha Kanté, and Andrew Twigg. 2008. Connectivity check in 3-connected planar graphs with obstacles. *Electron. Notes Discret. Math.* 31 (2008), 151–155. https://doi.org/10.1016/j.endm.2008.06.030

[9] Bruno Courcelle and Andrew Twigg. 2007. Compact Forbidden-Set Routing. In *Proceedings 24th Annual Symposium on Theoretical Aspects of Computer Science (STACS) (Lecture Notes in Computer Science, Vol. 4393)*. Springer, 37–48. https://doi.org/10.1007/978-3-540-70918-3_4

[10] Michal Dory and Merav Parter. 2021. Fault-Tolerant Labeling and Compact Routing Schemes. In *Proceedings of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*. 445–455. https://doi.org/10.1145/3465084.3467929

[11] Ran Duan, Yong Gu, and Hanlin Ren. 2021. Approximate Distance Oracles Subject to Multiple Vertex Failures. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2497–2516.

[12] Ran Duan and Seth Pettie. 2020. Connectivity Oracles for Graphs Subject to Vertex Failures. *SIAM J. Comput.* 49, 6 (2020), 1363–1396. https://doi.org/10.1137/17M1146610

[13] Martin Fürer and Balaji Raghavachari. 1994. Approximating the Minimum-Degree Steiner Tree to within One of Optimal. *J. Algor.* 17, 3 (1994), 409–423. https://doi.org/10.1006/jagm.1994.1042

[14] Mohsen Ghaffari and Merav Parter. 2016. MST in Log-Star Rounds of Congested Clique. In *Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*. 19–28. https://doi.org/10.1145/2933057.2933103

[15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*. 21–30.

[16] Taisuke Izumi, Yuval Emek, Tadashi Wadayama, and Toshimitsu Masuzawa. 2023. Deterministic Fault-Tolerant Connectivity Labeling Scheme with Adaptive Query Processing Time. In *Proceedings of the 42nd ACM Symposium on Principles of Distributed Computing (PODC)*. https://doi.org/10.48550/arXiv.2208.11459

[17] Arkady Kanevsky, Roberto Tamassia, Giuseppe Di Battista, and Jianer Chen. 1991. On-Line Maintenance of the Four-Connected Components of a Graph. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (FOCS)*. 793–801.

[18] Bruce M. Kapron, Valerie King, and Ben Mountjoy. 2013. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1131–1142.

[19] Karthik C. S. and Merav Parter. 2021. Deterministic Replacement Path Covering. In *Proceedings of the 32nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 704–723. https://doi.org/10.1137/1.9781611976465.44

[20] Neelesh Khanna and Surender Baswana. 2010. Approximate Shortest Paths Avoiding a Failed Vertex: Optimal Size Data Structures for Unweighted Graphs. In *Proceedings 27th Int'l Symposium on Theoretical Aspects of Computer Science (STACS)*. 513–524.

[21] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 2016. Higher Lower Bounds from the 3SUM Conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1272–1287. https://doi.org/10.1137/1.9781611974331.ch89

[22] Yaowei Long and Thatchaphol Saranurak. 2022. Near-Optimal Deterministic Vertex-Failure Connectivity Oracles. In *Proceedings 63rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 1002–1010. https://doi.org/10.1109/FOCS54457.2022.00098

[23] Hiroshi Nagamochi and Toshihide Ibaraki. 1992. A Linear-Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph. *Algorithmica* 7, 5&6 (1992), 583–596.

[24] Joseph Naor and Moni Naor. 1993. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. Comput.* 22, 4 (1993), 838–856.

[25] Jaroslav Nesetril, Eva Milková, and Helena Nesetrilová. 2001. Otakar Borůvka on minimum spanning tree problem —Translation of both the 1926 papers, comments, history. *Discret. Math.* 233, 1–3 (2001), 3–36. https://doi.org/10.1016/S0012-365X(00)00224-7

[26] Merav Parter and Asaf Petruschka. 2022. Õptimal Dual Vertex Failure Connectivity Labels. In *Proceedings of the 36th International Symposium on Distributed Computing (DISC) (LIPIcs, Vol. 246)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 32:1–32:19. https://doi.org/10.4230/LIPIcs.DISC.2022.32

[27] Merav Parter, Asaf Petruschka, and Seth Pettie. 2023. Connectivity Labeling and Routing with Multiple Vertex Failures. arXiv:2307.06276 [cs.DS]

[28] Seth Pettie and Longhui Yin. 2021. The Structure of Minimum Vertex Cuts. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP) (LIPIcs, Vol. 198)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 105:1–105:20. https://doi.org/10.4230/LIPIcs.ICALP.2021.105

[29] Michal Pilipczuk, Nicole Schirrmacher, Sebastian Siebertz, Szymon Torunczyk, and Alexandre Vigny. 2022. Algorithms and Data Structures for First-Order Logic with Connectivity Under Vertex Failures. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP) (LIPIcs, Vol. 229)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 102:1–102:18. https://doi.org/10.4230/LIPIcs.ICALP.2022.102

[30] Jan van den Brand and Thatchaphol Saranurak. 2019. Sensitive Distance and Reachability Oracles for Large Batch Updates. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 424–435. https://doi.org/10.1109/FOCS.2019.00034

[31] Oren Weimann and Raphael Yuster. 2013. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Transactions on Algorithms (TALG)* 9, 2 (2013), 14.

Received 09-NOV-2023; accepted 2024-02-11